Attorney Docket No. 06502.0369
Client Docket No. P5245
Express Mail No. EL396216928US

UNITED STATES PATENT APPLICATION

FOR

FLOATING POINT STATUS INFORMATION ACCUMULATION CIRCUIT

BY

GUY L. STEELE JR.

INCORPORATION BY REFERENCE

[001] Related U.S. Patent Application Serial No. _____, filed on even date herewith in the name of Guy L. Steele Jr. and entitled "Floating Point System That Represents Status Flag Information Within a Floating Point Operand," assigned to the assignee of the present application, incorporated by reference.

FIELD OF THE INVENTION

[002] The invention relates generally to systems and methods for performing floating point operations, and more particularly to systems and methods for accumulating floating point status information associated with a plurality of floating point values.

BACKGROUND OF THE INVENTION

[003] Digital electronic devices, such as digital computers, calculators, and other devices, perform arithmetic calculations on values in integer or "fixed point" format, in fractional or "floating point" format, or both. IEEE Standard 754 (hereinafter "IEEE Std. 754" or "the Standard"), published in 1985 by the Institute of Electrical and Electronic Engineers and adopted by the American National Standards Institute (ANSI), defines several standard formats for expressing values in floating point format and a number of aspects regarding behavior of computation in connection therewith. In accordance with IEEE Std. 754, a representation in floating point format comprises a plurality of binary digits, or "bits," having the structure

$$se_{msb} \cdots e_{lsb} f_{msb} \cdots f_{lsb}$$

where bit "s" is a sign bit indicating whether the entire value is positive or negative, bits "$e_{msb} \cdots e_{lsb}$" comprise an exponent field that represents the exponent "e" in unsigned binary biased format, and bits "$f_{msb} \cdots f_{lsb}$" comprise a fraction field that represents the fractional portion "f" in unsigned binary format ("msb" represents "most significant bit" and "lsb" represents "least significant bit"). The Standard defines two general formats, namely, a "single" format which comprises thirty-two bits, and a "double" format which comprises sixty-four bits. In the single format, there is one sign bit "s," eight bits "$e_{7...}e_0$" comprising the exponent field and twenty-three bits "$f_{22} \cdots f_0$" comprising the fraction field. In the double format, there is one sign bit "s," eleven bits "$e_{10} \cdots e_0$" comprising the exponent field and fifty-two bits "$f_{51} \cdots f_0$" comprising the fraction field.

[004] As indicated above, the exponent field of the floating point representation "$e_{msb} \cdots e_{lsb}$" represents the exponent "E" in biased format. The biased format provides a mechanism by which the sign of the exponent is implicitly indicated. In particular, the bits "$e_{msb...}e_{lsb}$" represent a binary encoded value "e" such that "e=E+bias." This allows the exponent E to extend from -126 to +127, in the eight-bit "single" format, and from -1022 to +1023 in the eleven-bit "double" format, and provides for relatively easy manipulation of the exponents in multiplication and division operations, in which the exponents are added and subtracted, respectively.

[005] IEEE Std. 754 provides for several different formats with both the single and double formats which are generally based on the bit patterns of the bits

"$e_{msb...}e_{lsb}$" comprising the exponent field and the bits $f_{msb}\cdots f_{lsb}$ comprising the fraction field. If a number is represented such that all of the bits "$e_{msb...}e_{lsb}$" of the exponent field are binary one's (i.e., if the bits represent a binary-encoded value of "255" in the single format or "2047" in the double format) and all of the bits $f_{msb}\cdots f_{lsb}$ of the fraction field are binary zeros, then the value of the number is positive or negative infinity, depending on the value of the sign bit "s". In particular, the value "v" is $v=(-1)^s\infty$, where "∞" represents the value of "infinity." On the other hand, if all of the bits "$e_{msb...}e_{lsb}$" of the exponent field are binary one's and if the bits $f_{msb}\cdots f_{lsb}$ of the fraction field are not all zero's, then the value that is represented is deemed "not a number," abbreviated in the Standard by "NaN."

[006] If a number has an exponent field in which the bits "$e_{msb...}e_{lsb}$" are neither all binary ones nor all binary zeros (i.e., if the bits represent a binary-encoded value between 1 and 254 in the single format or between 1 and 2046 in the double format), the number is said to be in a "normalized" format. For a number in the normalized format, the value represented by the number is $v=(-1)^s 2^{e-bias}(1.|f_{msb}\cdots f_{lsb})$, where " | " represents a concatenation operation. Effectively, in the normalized format, there is an implicit most significant digit having the value "one," so that the twenty-three digits in the fraction field of the single format, or the fifty-two digits in the fraction field of the double format, will effectively represent a value having twenty-four digits or fifty-three digits of precision, respectively, where the value is less than two, but not less than one.

[007] On the other hand, if a number has an exponent field in which the bits "$e_{msb}...e_{lsb}$" are all binary zeros, representing the binary-encoded value of "zero," and a fraction field in which the bits $f_{msb} \cdots f_{lsb}$ are not all zero, the number is said to be in a "denormalized" format. For a number in the denormalized format, the value represented by the number is $v = (-1)^s 2^{e-bias+1}(0.f_{msb...}f_{lsb})$. It will be appreciated that the range of values of numbers that can be expressed in the denormalized format is disjoint from the range of values of numbers that can be expressed in the normalized format, for both the single and double formats. Finally, if a number has an exponent field in which the bits "$e_{msb}...e_{lsb}$" are all binary zeros, representing the binary-encoded value of "zero" and a fraction field in which the bits $f_{msb} \cdots f_{lsb}$ are all zero, the number has the value "zero" (reference format 30). It will be appreciated that the value "zero" may be positive zero or negative zero, depending on the value of the sign bit.

[008] Generally, floating point units to perform computations whose results conform to IEEE Std. 754 are designed to generate a result in response to a floating point instruction in three steps:

[009] (a) In the first step, an approximation calculation step, an approximation to the absolutely accurate mathematical result (assuming that the input operands represent the specific mathematical values as described by IEEE Std. 754) is calculated that is sufficiently precise. This allows the accurate mathematical result to be summarized by a sign bit, an exponent (typically represented using more bits than are used for an exponent in the standard floating

point format), and some number "N" of bits of the presumed result fraction, plus a guard bit and a sticky bit. The value of the exponent will be such that the value of the fraction generated in step (a) consists of a "1" before the binary point and a fraction after the binary point. The bits are calculated so as to obtain the same result as the following conceptual procedure (which is impossible under some circumstances to carry out in practice): calculate the mathematical result to an infinite number of bits of precision in binary scientific notation, and in such a way that there is no bit position in the significand such that all bits of lesser significance are 1-bits (this restriction avoids the ambiguity between, for example, 1.100000...and 1.011111... as representations of the value "one-and-one-half"); then let the N most significant bits of the infinite significand be used as the intermediate result significand, let the next bit of the infinite significand be the guard bit, and let the sticky bit be "0" if and only if all remaining bits of the infinite significand are 0-bits (in other words, the sticky bit is the logical OR of all remaining bits of the infinite fraction after the guard bit).

[0010] (b) In the second step, a rounding step, the guard bit, the sticky bit, perhaps the sign bit, and perhaps some of the bits of the presumed significand generated in step (a) are used to decide whether to alter the result of step (a). For the rounding modes defined by IEEE Std. 754, this is a decision as to whether to increase the magnitude of the number represented by the presumed exponent and fraction generated in step (a). Increasing the magnitude of the number is done by adding "1" to the significand in its least significant bit position, as if the significand were a binary integer. It will be appreciated that, if the significand is all 1-bits, then

magnitude of the number is "increased" by changing it to a high-order 1-bit followed by all 0-bits and adding "1" to the exponent.

[0011] Regarding the rounding modes, it will be further appreciated that,

[0012] (i) if the result is a positive number, and

[0013] (a) if the decision is made to increase, effectively the decision has been made to increase the value of the result, thereby rounding the result up (i.e., towards positive infinity), but

[0014] (b) if the decision is made not to increase, effectively the decision has been made to decrease the value of the result, thereby rounding the result down (i.e., towards negative infinity); and

[0015] (ii) if the result is a negative number, and

[0016] (a) if the decision is made to increase, effectively the decision has been made to decrease the value of the result, thereby rounding the result down, but

[0017] (b) if the decision is made not to increase, effectively the decision has been made to increase the value of the result, thereby rounding the result up.

[0018] (c) In the third step, a packaging step, the result is packaged into a standard floating point format. This may involve substituting a special representation, such as the representation defined for infinity or NaN if an exceptional situation (such as overflow, underflow, or an invalid operation) was detected. Alternatively, this may involve removing the leading 1-bit (if any) of the fraction, because such leading 1-bits are implicit in the standard format. As another

alternative, this may involve shifting the fraction in order to construct a denormalized number. As a specific example, it is assumed that this is the step that forces the result to be a NaN if any input operand is a NaN. In this step, the decision is also made as to whether the result should be an infinity. It will be appreciated that, if the result is to be a NaN or infinity, any result from step (b) will be discarded and instead the appropriate representation will be provided as the result.

[0019] In addition, in the packaging step, floating point status information is generated, which is stored in a floating point status register. The floating point status information generated for a particular floating point operation includes indications, for example, as to whether

[0020] (i) a particular operand is invalid for the operation to be performed ("invalid operation");

[0021] (ii) if the operation to be performed is division, the divisor is zero ("division-by-zero");

[0022] (iii) an overflow occurred during the operation ("overflow");

[0023] (iv) an underflow occurred during the operation ("underflow"); and

[0024] (v) the rounded result of the operation is not exact ("inexact").

[0025] These conditions are typically represented by flags that are stored in the floating point status register separate from the result itself. The floating point status information can be used to dynamically control the operations in response to certain instructions, such as conditional branch, conditional move, and conditional trap instructions that may be in the instruction stream subsequent to the floating

point instruction. Also, the floating point status information may enable processing of a trap sequence, which will interrupt the normal flow of program execution. In addition, the floating point status information may be used to affect certain ones of the functional unit control signals that control the rounding mode. IEEE Std. 754 also provides for accumulating floating point status information from, for example, results generated for a series or plurality of floating point operations.

[0026] IEEE Std. 754 has brought relative harmony and stability to floating point computation and architectural design of floating point units. Moreover, its design was based on some important principles and rests on a sensible mathematical semantics that eases the job of programmers and numerical analysis. It also supports the implementation of interval arithmetic, which may prove to be preferable to simple scalar arithmetic for many tasks. Nevertheless, IEEE Std. 754 has some serious drawbacks, including:

[0027] (i) Modes (e.g., the rounding modes and traps enabled/disabled mode), flags (e.g., flags representing the status information stored in the floating point status register 25), and traps required to implement IEEE Std. 754 introduce implicit serialization issues. Implicit serialization is essentially the need for serial control of access (read/write) to and from globally used registers, such as the floating point status register 25. Under IEEE Std. 754, implicit serialization may arise between (1) different concurrent floating-point instructions and (2) between floating point instructions and the instructions that read and write the flags and modes. Furthermore, rounding modes may introduce implicit serialization because they are typically indicated as a global state, although in some microprocessor

architectures, the rounding mode is encoded as part of the instruction operation code, which will alleviate this problem to that extent. Thus, the potential for implicit serialization makes the Standard difficult to implement coherently and efficiently in today's superscalar and parallel processing architectures without loss of performance.

[0028] (ii) The implicit side effects of a procedure that can change the flags or modes can make it very difficult for compilers to perform optimizations on floating point code. As a result, compilers for most languages must assume that every procedure call is an optimization barrier in order to be safe.

[0029] (iii) Global flags, such as those that signal certain modes, make it more difficult to do instruction scheduling where the best performance is provided by interleaving instructions of unrelated computations. Instructions from regions of code governed by different flag settings or different flag detection requirements cannot easily be interleaved when they must share a single set of global flag bits.

[0030] (iv) Furthermore, traps have been difficult to integrate efficiently into architectures and programming language designs for fine-grained control of algorithmic behavior.

[0031] IEEE Std. 754 stores flag information as a global state and accumulates flag information from every operation, which makes it awkward and inefficient to accumulate flag status information from only a subset of executed operations, or to accumulate respective flag information separately and independently for a plurality of expressions, the execution of whose individual operations might be concurrent or interleaved in time.

[0032] In addition to the above drawbacks, existing computer architectures do not eliminate flags and trap enable bits as a global state, while supporting similar exception detection capabilities, even though existing computer architectures eliminate the rounding modes as a global state. This is typically done by statistically encoding the rounding mode as part of the instruction code. Examples of computer architectures that eliminate the rounding modes as a global state are demonstrated by the Digital Equipment Company ALPHA architecture, which partially eliminates the rounding modes, and Sun Microsystem's MAJC architecture, which completely eliminates the rounding modes.

[0033] Thus, there is a need for a system that avoids such problems when performing floating point operations and, in particular, when accumulating status information related to floating point operands.

SUMMARY OF THE INVENTION

[0034] Methods, systems, and articles of manufacture consistent with the present invention overcome these shortcomings with an accumulating floating point circuit that combines information associated with a plurality of floating point values. This circuit combines the value and status information associated with floating point values without the use or access to a separate floating point status register.

[0035] More particularly stated, a floating point flag combining circuit consistent with the present invention, as embodied and broadly described herein, includes an analysis circuit, which receives a plurality of floating point operands, and a result assembler. The analysis circuit analyzes the plurality of floating point

operands received and provides an indication of one or more predetermined formats in which the floating point operands are represented.

[0036] The analysis circuit provides an indication that may represent predetermined formats such as an overflow, underflow, not-a-number (NaN), infinity, normalized, and denormalized format. In more detail, the predetermined format may represent a +OV status, a -OV status, a +UN status, and a -UN status. The predetermined format may also comprise a plurality of bits indicative of a predetermined type of operand condition resulting in one of the formats, such as the NAN or infinity format.

[0037] The result assembler then, based on the indication from the analysis circuit, combines the plurality of floating point operands, in the various predetermined formats, to assemble an accumulated result in the NaN format.

[0038] In yet another aspect of the present invention, a method for combining floating point status information from a plurality of floating point operands consistent with an embodiment of the present invention is described. The method comprises receiving a plurality of floating point operands, each having encoded status flag information; analyzing the plurality of floating point operands to provide indications of one or more predetermined formats in which the floating point operands are represented; generating control signals based on the indications of the one or more predetermined formats; and assembling an accumulated result from the generated control signals and other input signals. The accumulated result represents a value and combines the encoded status flag information from each of the floating point operands.

[0039] Additional advantages of the invention will be set forth in part in the description which follows, and in part will be obvious from the description, or may be learned by practicing the invention. The advantages of the invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims.

[0040] It is to be understood that both the foregoing general description and the following detailed description are exemplary and exemplary only and are not restrictive of the invention, as claimed.

[0041] The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate embodiments of the invention and together with the description, serve to explain the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0042] Figure 1 is a functional block diagram of an exemplary floating point status information accumulation circuit constructed consistent with an exemplary embodiment of the present invention.

[0043] Figure 2 depicts exemplary formats for representations of floating point operands used by the floating point status information accumulation circuit depicted in Figure 1, and which is consistent with an exemplary embodiment of the present invention.

DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

[0044] Reference will now be made in detail to exemplary embodiments of the present invention, examples of which are illustrated in the accompanying

drawings. Wherever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts.

[0045] Related U.S. Patent Application Serial No. _____, which has previously been incorporated by reference, describes an exemplary floating point unit in which floating point status information is encoded in the representations of the results generated thereby. The exemplary floating point unit includes a plurality of functional units, including an adder unit, a multiplier unit, a divider unit, a square root unit, a maximum/minimum unit, a comparator unit and a tester unit, all of which operate under control of functional unit control signals provided by a control unit. The present application is directed to an exemplary combining or accumulation circuit that can be used as another functional unit or as part of the aforementioned functional units in the floating point unit described in related U.S. Patent Application Serial No. _____, to accumulate floating point status information from a plurality of floating point values that may comprise results from a plurality of floating point operations.

[0046] Figure 1 is a functional block diagram of an exemplary floating point status information accumulation circuit 10 constructed in accordance with an embodiment of the invention. Generally, the floating point status information accumulation circuit 10 receives two floating point operands and generates therefrom a result, as will be described below, that accumulates floating point status information embedded in the operands. The accumulation of the floating point status information may be conducted in an associative or commutative flag combining operation.

[0047] Accumulation circuit 10 then generates a result in the form of a floating point value, at least part of which comprises the accumulated floating point status information. Since the floating point status information is part of the floating point representation of the result, instead of being separate and apart from the result, there is no need to access any external circuitry (e.g., floating point status register). Thus, the implicit serialization that is required by maintaining the floating point status information separate and apart from the result can be advantageously obviated.

[0048] The floating point status information accumulation circuit 10 encodes the floating point status information in results that are generated in a plurality of exemplary formats, which will be illustrated in connection with Figure 2. Figure 2 depicts exemplary formats of floating point operands that the floating point status information accumulation circuit 10 may receive and of results that it may generate. With reference to Figure 2, seven exemplary formats are depicted, including a zero format 60, an underflow format 61, a denormalized format 62, a normalized non-zero format 63, an overflow format 64, an infinity format 65, and a not-a-number (NaN) format 66. The exemplary zero format 60 is used to represent the values "zero" or, more specifically, positive or negative zero, depending on the value of "s," the sign bit.

[0049] The exemplary underflow format 61 provides a mechanism by which the floating point status information accumulation circuit 10 can indicate that the result of a computation is an underflow. In the underflow format, the sign bit "s" indicates whether the result is positive or negative, the bits $e_{msb} \cdots e_{lsb}$ of the exponent

field are all binary zero's, and the bits $f_{msb} \cdots f_{lsb+1}$ of the fraction field, except for the least-significant bit, are all binary zero's. The least significant bit f_{lsb} of the fraction field is a binary one.

[0050] The exemplary denormalized format 62 and normalized non-zero format 63 are used to represent finite non-zero floating point values substantially along the lines of that described above in connection with IEEE Std. 754. In both formats 62 and 63, the sign bit "s" indicates whether the result is positive or negative. The bits $e_{msb} \cdots e_{lsb}$ of the exponent field of the denormalized format 62 are all binary zero's. However, the bits $e_{msb} \cdots e_{lsb}$ of the exponent field of the normalized non-zero format 63 are mixed one's and zero's, except that the exponent field of the normalized non-zero format 63 will not have a pattern in which bits $e_{msb} \cdots e_{lsb+1}$ are all binary ones and the least significant bit e_{lsb} is zero and all of the bits $f_{msb} \cdots f_{lsb}$ of the fraction field are all binary one's. In exemplary denormalized format 62, the bits $f_{msb} \cdots f_{lsb}$ of the fraction field are not all binary zero's.

[0051] The exemplary overflow format 64 provides a mechanism by which the floating point status information accumulation circuit 10 can indicate that the result of a computation is an overflow. In the overflow format 64, the sign bit "s" indicates whether the result is positive or negative, the bits $e_{msb} \cdots e_{lsb+1}$ of the exponent field are all binary ones, with the least-significant bit e_{lsb} being zero. The bits $f_{msb} \cdots f_{lsb}$ of the fraction field are all binary ones.

[0052] The exemplary infinity format 65 provides a mechanism by which the floating point status information accumulation circuit 10 can indicate that the result is infinite. In the infinity format 65, the sign bit "s" indicates whether the result is positive or negative, the bits $e_{msb} \cdots e_{lsb}$ of the exponent field are all binary ones, and the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field are all binary zero's. The five least-significant bits $f_{lsb+4} \cdots f_{lsb}$ of the fraction field are flags, which will be described below.

[0053] The exemplary NaN (not-a-number) format 66 provides a mechanism by which the floating point status information accumulation circuit 10 can indicate that the result is not a number. In the NaN format, the sign bits "s" can be any value, the bits $e_{msb} \cdots e_{lsb}$ of the exponent field are all binary ones, and the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field are not all binary zero's. The five least significant bits $f_{lsb+4} \cdots f_{lsb}$ of the fraction field are flags, which will be described below.

[0054] As noted above, in values represented in the exemplary infinity format 65 and the exemplary NaN format 66, the five low order bits $f_{lsb+4} \cdots f_{lsb}$ of the fraction field are flags. In one embodiment, the formats used with the floating point status information accumulation circuit 10 include the five flags that are defined by IEEE Std. 754. These flags include an invalid operation flag "n," and overflow flag "o," an underflow flag "u," a division-by-zero flag "z", and an inexact flag "x". For example, a value in the NaN format 66 in which both the overflow flag "o" and the division-by-zero flag "z" are set indicates that the value represents a result of a computation that involved an overflow (this from the overflow flag "o"), as well as an attempt to divide by zero (this from the division-by-zero flag "z").

[0055] In one embodiment, the flags may provide the same status information as provided by, for example, information stored in a floating point status register in a prior art floating point unit. In this embodiment, the information is provided as part of the result and stored therewith in registers in which the result is ultimately stored. Therefore, multiple instructions can be contemporaneously executed and the floating point status information that may be generated during execution of one instruction, when stored, will not over-write previously-stored floating point status information generated during execution of another instruction.

[0056] In another embodiment, values in the other formats can be indicated as being inexact based in part on the least-significant bit f_{lsb} of their fraction fields. In that embodiment, that particular bit operates as an inexact flag. Thus, the value will be indicated as being inexact if the particular bit, such as the f_{lsb}, has the value "one". Otherwise, the operand has an exact status.

[0057] With this background on the exemplary operand formats, the structure and operation of exemplary accumulation circuit 10 will be described in connection with Figure 1. Generally, embodiments of the invention provide an arrangement, in the form of the floating point status information accumulation circuit 10, that accumulates floating point status information, as represented by the flags within a plurality of floating point values. The floating point values from which floating point status information is accumulated may be results of, for example, successive floating point operations. In basic operation, the floating point status

information accumulation circuit 10 receives two operands and generates in response a value in the exemplary NaN format 66. If:

[0058] (A) both operands are values in the exemplary infinity format 65 or the exemplary NaN format 66,

[0059] (a) the flags of the result will be the flag-wise OR of the respective flags "n," "o," "u," "z," and "x" of the two operands, and

[0060] (b) if:

[0061] (i) both operands are in the exemplary NaN format 66, the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result will correspond to the corresponding bits of the operand that has the larger binary-encoded value,

[0062] (ii) one operand is in the exemplary NaN format 66 and the other operand is in the infinity format, the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result will correspond to the corresponding bits of the operand that is in the exemplary NaN format 66, and

[0063] (iii) otherwise (e.g., if both operands are in the exemplary infinity format 65), the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result will correspond to a predetermined bit pattern 0000000000000001;

[0064] (B) one operand is a value in the exemplary infinity format 65 or the exemplary NaN format 66, and the other operand is not in the exemplary infinity format 65, the exemplary NaN format 66, the exemplary overflow format 64, or the exemplary underflow format 61,

[0065] (a) flags "n," "o," "u," and "z" of the result will correspond to the respective flags of the operand that is in the NaN format 66 or infinity format 65,

[0066] (b) the inexact flag "x" of the result will be the logical OR of the inexact flags of the two operands, and

[0067] (c) if:

[0068] (i) the one operand is in the exemplary NaN format 66, the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result will correspond to the corresponding bits of the operand that is in the exemplary NaN format 66, and

[0069] (ii) otherwise (e.g., if the one operand is in the infinity format 65), the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result will correspond to a predetermined bit pattern 0000000000000001;

[0070] (C) neither operand is a value in the exemplary infinity format 65, the exemplary NaN format 66, the exemplary overflow format 64 or the exemplary underflow format 61,

[0071] (a) all flags "n," "o," "u," and "z" of the result will be clear,

[0072] (b) the inexact flag "x" of the result will be the logical OR of the inexact flags of the two operands, and

[0073] (c) the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result will correspond to a predetermined bit pattern 0000000000000000001;

[0074] (D) neither operand is a value in the exemplary infinity format 65 or the exemplary NAN format 66, but at least one operand is in the exemplary overflow format 64 or the exemplary underflow format 61,

[0075] (a) flags "n" and "z" of the result will be clear,

[0076] (b) overflow flag "o" of the result will be set if either operand is in the overflow format 64,

[0077] (c) underflow flag "u" of the result will be set if either operand is in the underflow format 61,

[0078] (d) the inexact flag "x" of the result will be the logical OR of the inexact flags of the two operands, and

[0079] (e) the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result will correspond to a predetermined bit pattern 0000000000000000001;

[0080] (E) one operand is a value in the exemplary infinity format 65 or the exemplary NaN format 66, and the other operand is in the exemplary overflow format 64:

[0081] (a) flags "n," "u," and "z" of the result will correspond to the respective flags of the operand that is in the NaN format 66 or the infinity format 65,

[0082] (b) the inexact flag "x" of the result will be the logical OR of the inexact flags of the two operands,

[0083] (c) the "o" flag of the result will be set, and

[0084] (d) if:

[0085] (i) the one operand is in the exemplary NaN format 66, the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result will correspond to the corresponding bits of the operand that is in the exemplary NaN format 66, and

[0086] (ii) otherwise (e.g., if the one operand is in the exemplary infinity format 65) the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result will correspond to a predetermined bit pattern 0000000000000001; and

[0087] (F) one operand is a value in the exemplary infinity format 65 or the exemplary NaN format 66, and the other operand is in the exemplary underflow format 64:

[0088] (a) flags "n," "o," and "z" of the result will correspond to the respective flags of the operand that is in the NaN format 66 or the infinity format 65,

[0089] (b) the inexact flag "x" of the result will be the logical OR of the inexact flags of the two operands,

[0090] (c) the "u" flag of the result will be set, and

[0091] (d) if:

[0092] (i) the one operand is in the exemplary NaN format 66, the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result will correspond to the corresponding bits of the operand that is in the exemplary NaN format 66, and

[0093] (ii) otherwise (e.g., if the one operand is in the exemplary infinity format 65) the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result will correspond to a predetermined bit pattern 000000000000000001.

[0094] In all the above mentioned cases, the sign bit "s" of the result will correspond to the logical OR of the sign bits "s" of the operands.

[0095] It will be appreciated that, if one operand is in the exemplary overflow format 64 and the other operand is in the exemplary underflow format 61 (reference item (D) above), both the overflow flag "o" and the underflow flag "u" of the result will be set.

[0096] As noted above, in all cases the result will be in the exemplary NaN format 66, regardless of the formats of the operands, and, if neither operand is in the NaN format 66, the floating point status information accumulation circuit 10 generates a result (also called an accumulated result) in the exemplary NaN format 66 with the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field having the bit pattern 000000000000000001. It will be appreciated that this bit pattern has the lowest possible binary-encoded value. In this case, if the result is subsequently used as an operand in the floating point status information accumulation circuit 10 and if the other operand is a value in the exemplary NaN format 66 in which the most

significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field has a higher value, the floating point status information accumulation circuit 10 will generate a result in which the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field corresponds to the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the other operand (reference item (A) above).

[0097] In addition, it will be appreciated that, since the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of values in the infinity format 65 have the bit pattern 000000000000000000, the binary-encoded value of that portion of the fraction field will be less than the binary-encoded value of the most significant bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of any value in the exemplary NaN format 66.

[0098] With reference to Figure 1, the illustrated embodiment of the floating point status information accumulation circuit 10 has two operand buffers 11A and 11B, respective operand analysis circuits 12A and 12B, a decision circuit 13, and a result assembler 14. The operand buffers 11A and 11B receive and store respective operands from, for example, a set of registers (not shown) in a conventional manner. Each operand analysis circuit 12A and 12B analyzes the operand in the respective buffer 11A and 11B, and generates signals providing information relating to the respective operands, which are provided to the decision circuit 13. The signals provided by the respective operand analysis circuit 12A and 12B essentially indicate whether the format of the respective operand is in one of the exemplary formats, such as the underflow format 61, the overflow format 64, the infinity format 65, or the NaN format 66. The decision circuit 13 receives the signals from the operand analysis circuits 12A and 12B and generates control signals that

control the result assembler 14 in assembling the result. The result assembler 14 receives information from a number of sources, including the operand buffers 11A and 11B, and several predetermined value stores as described below. Under control of control signals from the decision circuit 13 and signals from the operand analysis circuits 12A and 12B, result assembler 14 assembles the result, which is provided on a result bus 17. Result bus 17, in turn, may deliver the result to any convenient destination, such as a register in a register set (not shown), for storage or other use.

[0099] As noted above, each operand analysis circuit 12A and 12B analyzes the operand in the respective buffer 11A and 11B and generates signals providing information relating to the respective operands. In the illustrated embodiment, each exemplary operand analysis circuit 12A and 12B comprises a number of comparators, including:

[00100] (i) a comparator 20A and 20B that generates an asserted signal if the bits $e_{msb} \cdots e_{lsb}$ of the exponent field of the operand in respective buffer 11A and 11B are all binary one's, which will be the case if the operand is in the exemplary infinity format 65 or the exemplary NaN format 66;

[00101] (ii) a comparator 21A and 21B that generates an asserted signal if the bits $e_{msb} \cdots e_{lsb+1}$ of the exponent field of the operand in the respective buffer 11A and 11B are all binary one's, and the bit e_{lsb} is a binary zero, which will be the case if the operand is in the exemplary overflow format 64;

[00102] (iii) a comparator 22A and 22B that generates an asserted signal if the bit $e_{msb} \cdots e_{lsb}$ of the exponent field of the operand in respective buffer 11A and 11B are all binary zero's, which will be the case if the operand is in one of the exemplary formats, such as the zero format 60, underflow format 61, or denormalized format 62;

[00103] (iv) a comparator 30A and 30B that generates an asserted signal if the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in the respective buffer 11A and 11B are all binary one's, which may be the case if the operand is in one of the exemplary formats, such as the denormalized format 62, normalized non-zero format 63, overflow format 64, or NaN format 66;

[00104] (v) a comparator 31A and 31B that generates an asserted signal if the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in the respective buffer 11A and 11B are all binary zero's, which may be the case if the operand is in one of the exemplary formats, such as the zero format 60, underflow format 61, denormalized format 62, normalized non-zero format 63, or infinity format 65;

[00105] (vi) a comparator 32A and 32B that generates an asserted signal if the bits $f_{lsb+4} \cdots f_{lsb}$ of the fraction field of the operand in the respective buffer 11A and 1B are all binary one's, which may be the case if the operand is in one of the exemplary formats, such as the denormalized format 62, normalized non-zero format 63, overflow format 64, or if all of the flags "n," "o," "u," "z," and "x" are set in the infinity format 65 or NaN format 66; and

[00106] (vii) a comparator 33A and 33B that generates an asserted signal if the bits $f_{lsb+4} \cdots f_{lsb+1}$ of the fraction field of the operand in the respective buffer 11A and 11B are all binary zero's and if the bit f_{lsb} of the fraction field is a binary "one," which may be the case if the operand is in one of the exemplary formats, such as the underflow format 61, the denormalized format 62, the normalized non-zero format 63, or if the flags "n," "o," "u," and "z" are clear and the flag "x" is set, in the infinity format 65 or NaN format 66.

[00107] In the illustrated embodiment, each operand analysis circuit 12A and 12B also includes combinatorial logic elements that receive selected signals from the comparators and generate asserted signals to provide indications as to certain characteristics of the respective operand, including:

[00108] (i) an AND gate 36A and 36B that may generate an asserted signal if comparators 22A, 22B, 31A, 31B, 33A, and 33B are all generating asserted signals, which may be the case if the bits $e_{msb} \cdots e_{lsb}$ of the exponent field of the operand in the respective buffer have the bit pattern 00000000 and bits $f_{msb} \cdots f_{lsb}$ of the fraction field of the operand in the respective operand buffer 11A and 11B have the bit pattern 00000000000000000000001 (it will be appreciated that if the AND gate 36A and 36B is generating an asserted signal, the operand in the respective operand buffer is in the exemplary underflow format 31);

[00109] (ii) an AND gate 37A and 37B that may generate an asserted signal if comparators 21A, 21B, 30A, 30B, 32A, and 32B are all generating asserted signals, which may be the case if the bits $e_{msb} \cdots e_{lsb}$ of the exponent field of the

operand in the respective buffer have the bit pattern 11111110 and bits $f_{msb} \cdots f_{lsb}$ of the fraction field of the operand in the respective operand buffer 11A and 11B have the bit pattern 11111111111111111 (it will be appreciated that if the AND gate 37A or 37B is generating an asserted signal, the operand in the respective operand buffer is in the exemplary overflow format 64); and

[00110] (iii) a NAND gate 38A and 38B that may generate an asserted signal if comparator 20A and 20B is generating an asserted signal and comparator 31A and 31B is generating a negated signal, which may be the case if the bits $e_{msb} \cdots e_{lsb}$ of the exponent field of the operand in the respective buffer have the bit pattern 11111111 and at least one bit $f_{msb} \cdots f_{lsb}$ of the fraction field of the operand in the respective operand buffer 11A and 11B has the value "one" (it will be appreciated that if the NAND gate 38A and 38B is generating an asserted signal, the operand in the respective operand buffer is in the exemplary NaN format 66).

[00111] As noted above, the decision circuit 13 receives the signals from the operand analysis circuits 12A and 12B and generates control signals that control the result assembler 14 in assembling the result. In the illustrated embodiment, the decision circuit 13 comprises:

[00112] (i) a comparator 40 that generates an asserted signal if the binary-encoded value of the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in operand buffer 11A is greater than the binary-encoded value of the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in operand buffer 11B; otherwise, a negated signal is generated by comparator 40;

[00113] (ii) a NOR gate 41 that generates an asserted signal if the comparator 40 is generating an asserted signal or if the NAND gate 38B is generating a negated signal;

[00114] (iii) a NOR gate 42 that generates an asserted signal if either the comparator 40 or the NAND gate 38A is generating a negated signal;

[00115] (iv) an AND gate 43 that generates an asserted signal if both the NAND gate 38A and the NOR gate 41 are generating asserted signals;

[00116] (v) an AND gate 44 that generates an asserted signal if both the NAND gate 38B and the NOR gate 42 are generating asserted signals;

[00117] (vi) a NAND gate 45 that generates an asserted signal if both the NAND gate 38A and the NAND gate 38B are generating negated signals;

[00118] (vii) an OR gate 46 that generates an asserted signal if either AND gate 36A or AND gate 36B is generating an asserted signal; and

[00119] (viii) an OR gate 47 that generates an asserted signal if either AND gate 37A or AND gate 37B is generating an asserted signal.

[00120] Regarding exemplary decision circuit 13, it will be further appreciated that:

[00121] (1) AND gate 43 (reference item (iv) above) will generate an asserted signal if both:

[00122] (i) the operand in operand buffer 11A is in the exemplary NaN format 66, and

[00123] (ii) either:

[00124] (a) the operand in operand buffer 11B is not in the exemplary NaN format 66, or

[00125] (b) the binary-encoded operand of the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in operand buffer 11A is greater than the binary-encoded operand of the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in operand buffer 11B;

[00126] (2) AND gate 44 (reference item (v) above) will generate an asserted signal if both:

[00127] (i) the operand in operand buffer 11B is in the exemplary NaN format 66, and

[00128] (ii) either:

[00129] (a) the operand in operand buffer 11A is not in the exemplary NaN format 66, or

[00130] (b) the binary-encoded operand of the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in operand buffer 11A is not greater than (i.e., is less than or equal to) the binary-encoded operand of the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in operand buffer 11B;

[00131] (3) AND gate 45 (reference item (vi) above) will generate an asserted signal if both:

[00132] (i) the operand in operand buffer 11A is not in the exemplary NaN format 66, and

[00133] (ii) the operand in operand buffer 11B is not in the exemplary NaN format 66;

[00134] (4) OR gate 46 (reference item (vii) above) will generate an asserted signal if the operand in either operand buffer 11A or 11B is in the exemplary underflow format 61; and

[00135] (5) OR gate 47 (reference item (viii) above) will generate an asserted signal if the operand in either operand buffer 11A or 11B is in the exemplary overflow format 64.

[00136] As noted above, the result assembler 14 receives information from a number of sources, including the operand buffers 11A and 11B, and several predetermined value stores. Under the control of control signals from the decision circuit 13 and signals from the operand analysis circuits 12A and 12B, result assembler 14 assembles the result, which is provided on the result bus 17. In one embodiment, the result assembler 14 may assemble the result in four segments, including a sign segment that represents the sign bit of the result, an exponent segment that represents the exponent field of the result, a high-order fraction segment that represents the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result, and a low-order fraction segment that represents the five least-significant bits $f_{lsb+4} \cdots f_{lsb}$ of the result. It will be appreciated that the low-order fraction segment corresponds to the flags "n," "o," "u," "z," and "x."

[00137] More specifically and in accordance with the illustrated embodiment in Fig. 1, the exemplary result assembler 14 may be comprised of four elements,

including an OR gate 50, a predetermined value store 51, a selector 52 and a combiner circuit 53. The OR gate 50 receives signals from the operand buffers 11A and 11B representative of the sign bits of the respective operands and generates an output signal representative of the logical OR of the respective signals. The output signal generated by the OR gate 50 is coupled onto the result bus 17 as the sign segment of the result.

[00138] The predetermined value store 51 stores a value that corresponds to the bit pattern 11111111, which corresponds to the bit pattern of the bits $e_{msb} \cdots e_{lsb}$ comprising the exponent field of values in the exemplary NaN format 66. The predetermined value store 51 couples signals representative of the bit pattern stored therein onto the result bus as the exponent segment of the result.

[00139] The selector 52 couples high-order fraction field signals representative of the high-order fraction field bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the result to the result bus 17. The selector 52 receives three sets of high-order fraction field value signals, namely, signals representative of bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in buffer 11A, signals representative of bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in buffer 11B, and a set of signals representative of a predetermined high-order fraction field bit pattern 0000000000000001. In addition, the selector 52 receives control signals from the decision circuit 13, specifically from AND gates 43, 44, and 45. Each control signal is associated with one of the sets of high-order fraction field value signals. In the illustrated embodiment, it will be appreciated that exactly one of the signals from AND gates

43, 44, and 45 will be asserted at any point in time and, when a control signal is asserted, the selector 52 will couple high-order fraction field value signals associated with the asserted control signal to the result bus 17 as the high-order fraction segment. Accordingly, if the signal from the AND gate 43 is asserted, the selector 52 will couple signals representative of bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in buffer 11A to the result bus as the high-order fraction field segment. As noted above (reference items (1) above), this will occur if both:

[00140] (i) the operand in operand buffer 11A is in the exemplary NaN format 66, and

[00141] (ii) either:

[00142] (a) the operand in operand buffer 11B is not in the exemplary NaN format 66, or

[00143] (b) the binary-encoded operand of the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in operand buffer 11A is greater than the binary-encoded operand of the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in operand buffer 11B.

[00144] On the other hand, if the signal from the AND gate 44 is asserted, the selector 52 will couple signals representative of bits $f_{msb} \cdots f_{lsb+5}$ of the fractional field of the operand in buffer 11B to the result bus as the high-order fraction field segment. As noted above (reference item (2) above), this will occur if both:

[00145] (i) the operand in operand buffer 11B is in the exemplary NaN format 66, and

[00146] (ii) either:

[00147] (a) the operand in operand buffer 11A is not in the exemplary NaN format 66, or

[00148] (b) the binary-encoded operand of the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in operand buffer 11A is not greater than (i.e., is less than or equal to) the binary-encoded operand of the bits $f_{msb} \cdots f_{lsb+5}$ of the fraction field of the operand in operand buffer 11B.

[00149] Finally, if the signal from the AND gate 46 is asserted, the selector 52 will couple signals representative of the predetermined high-order fraction field bit pattern 0000000000000001 to the result bus as the high-order fraction field segment. As noted above (reference item (3) above), this will occur if both:

[00150] (i) the operand in operand buffer 11A is not in the exemplary NaN format 66, and

[00151] (ii) the operand in operand buffer 11B is not in the exemplary NaN format 66.

[00152] The combiner 53 couples low-order fraction field value signals representative of the low-order fraction field bits $f_{lsb+4} \cdots f_{lsb}$ of the fraction field of the result to the result bus 17. The combiner 53 receives signals representative of bits $f_{lsb+4} \cdots f_{lsb}$ of the fraction field of the operand in buffer 11A, signals representative of bits $f_{lsb+4} \cdots f_{lsb}$ of the fraction field of the operand in buffer 11B, and signals from OR gates 46 and 47. In addition, the combiner 53 receives signals from the comparators 20A and 20B. If the comparator 20A is generating an asserted signal,

which may be the case if the operand in operand buffer 11A is in the exemplary infinity format 65 or the exemplary NaN format 66, the least-significant bits $f_{lsb+4} \cdots f_{lsb}$ of the operand buffer 11A will contribute to the least-significant bits $f_{lsb+4} \cdots f_{lsb}$ of the result. Similarly, if the comparator 20B is generating an asserted signal, which may be the case if the operand in operand buffer 11B is in the exemplary infinity format 65 or the exemplary NaN format 66, the least-significant bits $f_{lsb+4} \cdots f_{lsb}$ of the operand in operand buffer 11B will contribute to the least-significant bits $f_{lsb+4} \cdots f_{lsb}$ of the result. The signal from the OR gate 46 will be asserted, if the operand in either operand buffer 11A or 11B is in the exemplary underflow format 61, and will contribute to the bit f_{lsb+2} of the result, which corresponds to the underflow flag "u." Similarly, the signal from the OR gate 47 will be asserted if the operand in either operand buffer 11A or 11B is in the overflow format 64 and will contribute to the bit f_{lsb+3} of the result, which corresponds to the overflow flag "o." Except in the embodiment described above in which the least-significant bit f_{lsb} of the fraction field of values in all formats comprise inexact flags, the least-significant bits $f_{lsb+4} \cdots f_{lsb}$ of values in other formats will not contribute to the bits $f_{lsb+4} \cdots f_{lsb}$ of the fraction field of the result. In the embodiment in which the least-significant bit f_{lsb} of the fraction field of values in all formats comprise inexact flags, the least-significant bit f_{lsb} of values in all formats will contribute to the bit f_{lsb} of the fraction field of the result.

[00153] In the following, it will be assumed that the floating point status information accumulation unit 10 comprises the embodiment in which the least-significant bit f_{lsb} of the fraction field of values in all formats comprise inexact flags. In that embodiment, the combiner 53 comprises a plurality of OR gates 54(0) through 54(4). OR gate 54(0) receives signals representative of the least-significant bits f_{lsb} of the fraction field of both operand buffers 11A and 11B and couples an output signal representative of the OR thereof onto the result bus 17 as the least-significant bit f_{lsb} of the result. Since the result is in the exemplary NaN format 66, the signal provided by the OR gate 54(0) is representative of the inexact flag "x."

[00154] Each of the other OR gates 54(1) through 54(4) (generally identified by reference numeral 54(i)) is associated with two AND gates 55(i)(A) and 55(i)(B). Each AND gate 55(i)(A) receives, as one input, a signal representative of the bit f_{lsb+i} of the fraction field of the operand in operand buffer 11A; and as its other input, the signal generated by comparator 20A. If the signal generated by comparator 20A is asserted, which may be the case if the operand in operand buffer 11A is in the exemplary infinity format 65 or the exemplary NaN format 66, the AND gate 55(i)(A) is energized to couple the signal representative of the bit f_{lsb+i} of the fraction field of the operand in operand buffer 11A to its output. The output of each AND gate 55(i)(A) is coupled to one input of the respective OR gate 54(i).

[00155] Similarly, each AND gate 55(i)(B) receives, as one input, a signal representative of the bit f_{lsb+i} of the fraction field of the operand in operand buffer 11B; and as its other input, the signal generated by comparator 20B. If the signal

generated by comparator 20B is asserted, which may be the case if the operand in operand buffer 11B is in the exemplary infinity format 65 or the exemplary NaN format 66, the AND gate 55(i)(B) is energized to couple the signal representative of the bit f_{lsb+i} of the fraction field of the operand in operand buffer 11B to its output. The output of each AND gate 55(i)(B) is coupled to one input of the respective OR gate 54(i).

[00156] OR gates 54(i) receive the signals provided by respective AND gates 55(i)(A) and 55(i)(B); and, in the case of OR gates 54(2) and 54(3), signals provided by OR gates 46 and 47. OR gates 54(i) couple output signals representative of the OR thereof onto the result bus 17 as respective bit f_{lsb+i} of the result. Accordingly, if the operand in respective operand buffer 11A and 11B is in the exemplary infinity format 65 or the exemplary NaN format 66, the least-significant bits $f_{lsb+4} \cdots f_{lsb+1}$ of its fraction field, which, as noted above, comprise flags "n," "o," "u," and "z," will contribute to the bits $f_{lsb+4} \cdots f_{lsb+1}$ of the fraction field of the result. In addition, since the signals provided by OR gates 46 and 47 are used by the OR gates 54(2) and 54(3) in generating their output signals, the bits f_{lsb+2} and f_{lsb+3} of the fraction field of the result, which correspond to the "u" and "o" flags, respectively, also indicate whether the operand in operand buffer 11A and 11B is in the respective exemplary underflow format 61 or overflow format 64.

[00157] The embodiments of the invention provide a number of advantages. In particular, the accumulation circuit 10 is provided in a system in which floating point status information associated with a value in a floating point representation is

embedded or encoded in the representation. The arrangement in the form of a floating point status information accumulation circuit 10 facilitates the accumulation of floating point status information from a plurality of floating point values in another floating point value. This advantageously obviates the necessity of providing separate floating point status registers to store either the floating point status information of the plurality of floating point values or the accumulated floating point status information.

[00158] One of ordinary skill in the art will recognize that other formats and bit patterns could be used to represent the floating point operand formats without departing from the principles of the present invention. One of ordinary skill in the art will also recognize that the floating point status information contained in the operands could easily be represented by other bit combinations (not shown) without departing from the principles of the present invention. For example, more or fewer bits could be used, a subset or superset of the exemplary status bits could be used, or the most significant bits of an operand (or some other subset of bits) could be used to indicate the floating point status information, instead of the least significant bits illustrated. As a further example, an alternative embodiment could deliver a status-containing result represented in more bits, or fewer bits, than the number of bits used to represent one of the operands. As still another example, an alternative embodiment might accept one operand in the form of a floating point number with embedded flag status information, and accept another operand or produce a result in the form of some other data structure with embedded flag status information without departing from the principles of the present invention.

[00159] It will be appreciated that a system in accordance with the invention can be constructed in whole or in part from special purpose hardware or a general purpose computer system, or any combination thereof, any portion of which may be controlled by a suitable program. Any program may in whole or in part comprise part of or be stored on the system in a conventional manner, or it may in whole or in part be provided into the system over a network or other mechanism for transferring information in a conventional manner. In addition, it will be appreciated that the system may be operated and/or otherwise controlled by means of information provided by an operator using operator input elements (not shown), which may be connected directly to the system or which may transfer the information to the system over a network or other mechanism for transferring information in a conventional manner.

[00160] It will also be appreciated that the invention may be practiced in an electrical circuit comprising discrete electronic elements, packaged or integrated electronic chips containing logic gates, a circuit utilizing a microprocessor, or on a single chip containing electronic elements or microprocessors. It may also be provided using other technologies capable of performing logical operations such as, for example, AND, OR, and NOT, including but not limited to mechanical, optical, fluidic, and quantum technologies. In addition, the invention may be practiced within a general purpose computer or in any other circuits or systems as are known by those skilled in the art.

[00161] The foregoing description has been limited to a specific embodiment of this invention. It will be apparent, however, that various variations and

modifications may be made to the invention, with the attainment of some or all of the advantages of the invention. It is the object of the appended claims to cover these and such other variations and modifications as come within the true spirit and scope of the invention.